Argonne NATIONAL LABORATORY

# Learning Python

## Getting results for beamlines and scientific programming

7. Basic Python: provided functions and modules

The Advanced Photon Source is an Office of Science User Facility operated for the U.S. Department of Energy Office of Science by Argonne National Laboratory

U.S. DEPARTMENT OF ENERGY | Office of Science

# Outline of topics to be covered

1. Built-in routines
2. Standard modules
3. Accessing modules
4. Other Python modules
5. The import search path

# What can Python do out of the box?

Python has many built in functions that are part of the basic language. In addition, there are many other useful capabilities provided as routines that are included in modules (libraries) that are provided as part of the language. The number and scope of these modules increase with each release.

In addition, there are many available modules for Python that are available from other sources for tasks such as graphics, GUIs, math and much more

# Built-in routines

There are 80 built-in functions in Python. Some we have seen, such as range(), len(), complex() and list(); others are not commonly used. The full set is documented in http://docs.python.org/library/functions.html. Below are some of my favorites:

- abs(x) – absolute value
- any(list) – True if one or more in list is true
- all(list) – True if all in list are true
- dir(object) – returns a list of functions defined for the object
- help(object) – provides docstring info on object
- max(list) – max value in list
- min(list) – min value in list
- open(file,mode) – open a file for reading or writing
- pow(x,y) – computes x**y
- raw_input(prompt) – obtains input from stdin (usually a DOS/ terminal window)
- round(x,n) -- round x to n decimal places (0 if n is omitted)
- sorted(list) – sort the objects in list
- zip(list1,list2,…) – built a new list with items from each list

# Standard Modules

Python supplies ~200 modules with the basic install of the language. A few are platform-specific but most are not. The full list can be found at http://docs.python.org/library/index.html but here are a few:

- math: mathematical functions
- os.path: manipulate directory names
- sys: hardware/OS-specific interfaces
- glob: file name searching
- os: interface to operating system functions
- re: regular expressions
- pickle: convert Python objects to strings for convenient storage
- time: access to time of day and CPU clocks
- datetime: time computations
- email: e-mail & attachment creation
- pdb: debugger
- subprocess: create blocking and async subprocesses
- pydoc: generate module documentation
- timeit: compute CPU use for a code snippet

# Using Standard Modules

Before a routine in a module can be accessed the module needs to be “loaded” into the currently running code. For standard modules this is done with an import statement:

```
import sys
```

When this is done, variables and routines in the module can be accessed by prepending the name of the module:

```
print sys.platform
sys.exit()
sys.path.append('./libs')
```

When you import a module, the code in that module is run, defining functions and variables.

Note that an import statement is needed in every module you write for access to that module. However, the code in the imported module is run only once.

# Hierarchical packages

- In many cases complex packages can be partitioned into levels (sub-packages)
- For example,
  - os.path is a section of the os package that deals with path manipulations
  - scipy.special contains commonly defined (aka “special”) functions

- You can load a sub-package only if you don’t need the parent:
  - `import os.path`

  Or
  - `import os`

  Both give access to the os.path module

- In some cases sub-packages are not loaded unless asked for explicitly:
  - `import scipy.special`

  This is the only way to access this module!

# Other versions of the import statement

When a module has a long name, it may be more convenient to assign a shorter name to that module. This can be done in two ways

```
import PackageWithALongName as PWLN
```

```
import PackageWithALongName
PWLN = PackageWithALongName
```

The later mechanism defines both PackageWithALongName and PWLN

It is common to use certain abbreviations, for example:

```
import numpy as np
```

# The from syntax for module imports

An alternate syntax for import allows modules to be imported and used without the prefix:

```
from sys import exit, platform
print platform
exit()
```

- It is also possible to import everything in a module in this manner:

```
from sys import *
```

- *I would recommend that you use the former syntax rarely and the latter (*from <module> import **) never. Why: It is much easier to understand and maintain code where functions and variables are prefixed by the name of the package.*

# Other modules

- There are many other modules available for Python; in fact web page http://pypi.python.org/ has an index to >14,000 of them. Externally maintained modules are called packages. It is sometimes easy and sometimes difficult to install a package. Bear in mind that adopting a package may make code non-portable, since not all packages run on all platforms.

Later in these lectures a few packages that are of great value will be presented:

- matplotlib: a plotting package
- wxPython: a portable GUI system based on wxwidgets
- numpy and scipy: packages that support rapid mathematical and scientific computation
- ca_util for EPICS (local to APS)

- It should be noted that there are many other packages out there and in the cases of matplotlib and wxPython, there may well be better choices.

# Looking for modules in all the wrong places

- Usually when modules are installed, they are placed in standard locations for the language and can be found immediately by an import statement. This may not be the case for modules that you wish to use that you or someone else has written.

- Modules in the same directory as the current script are usually located. However, if you wish to place the modules somewhere else, you need to tell the Python interpreter where to look. The interpreter searches through the Python path which is defined in sys.path (as a list) until it finds a matching path

```
import sys
sys.path.append(<path>)
```

or

```
sys.path.insert(0,<path>)
```

- The latter version makes <path> the first place to be searched.

# Homework

- Search the Python standard functions to find out how to convert an integer to its hexadecimal representation. Confirm that 255 (base 10) is FF (base 16).

- Find the module in the Python standard library that is used to handle times and dates
  - Write Python code that prints the current date and time. (Hint, since this is potentially a bit confusing, you will need to use a function named now() in class that is defined within this module.)
  - Print the date/time [value you get from now()] using str() and repr() to format it.